Motivation

- It is easy to write a subsumption checker for, e.g., ALC.
- It is also easy to extend this to a small TBox/Abox system.
- On top of that we need:
  1. a graphical user interface
  2. an application interface with the right kind of services
  3. an efficient implementation of the basic reasoning services
- Possible areas of optimization:
  1. Classification
  2. Subsumption test
  3. Preprocessing the KB

Classification: Sorting in Partial Orders

- Compute the hierarchy induced by the subsumption relationship (similar to sorting)
- Also important in other contexts
- Incremental creation:
  1. Find immediate subsumers and subsumees in hierarchy
  2. Place new concept at this place (and delete superfluous links)
Worst-Case Lower Bound for Classification

- Count number of subsumption tests
- What is the worst case?
  ~~~~ Worst case: All concepts are unrelated
  → \( O(n^2) \) for \( n \) being the number of concept names.
- Analytical comparison (for the average case) between different methods is probably difficult because it is unknown, how many partial orders exist for a given number of elements
  ~~~~ Empirical comparisons are the best we can hope for

Method 0: Brute force

- The brute-force method. For each new concept \( C \):
  - For each concept \( D \) in the hierarchy, check whether \( C \) subsumes \( D \) and whether \( D \) subsumes \( C \).
  - Compute from those the immediate subsumers and subsumees
  ~~~~ \( 2n \) tests for each new concept
  → \( \sum_{i=0}^{n-1} 2i = n \times (n - 1) \) for the whole KB

Method 1: Simple Traversal

- Top search: Traverse hierarchy top down (e.g. depth-first), and try to identify direct subsumers by checking subsumption and mark subsumption failures.
- Bottom search: dual, from bottom element

Method 2: Smart Traversal

- Exploit information from previous subsumption tests
- Use information about positive results in the top search path

subsumption test:
- positive
- negative
- no test

positive result inferred

- Use information about negative results in the top search path

subsumption test:
- positive
- negative
- no test

negative information inferred
Three Alternatives

Note: Smart traversal is provably as good as simple traversal (counting the number of subsumption tests) and the additional overhead is negligible (0.5% of the subsumption costs in a typical KB).

Alternatives:
1. Depth-first search using only positive information
2. Breadth-first search using only negative information
3. Depth-first search using both negative and positive information

Our results: Breadth-first search is best. Using only positive information does not help much.

Number of Subsumption Tests Relative to “Brute Force” Method

In 1992, we evaluated the effects of different techniques.
On real KBs, the book keeping overhead turned out to be relatively low compared with the time for subsumption checking.
We measured the effects on randomly generated KBs modeled after the structure of real KBs.
### Results for Chain Insertion

<table>
<thead>
<tr>
<th>Type of KB</th>
<th>No. of nodes</th>
<th>Average type</th>
<th>Average no. of pred. &amp; succ.</th>
<th>Breadth</th>
<th>Depth</th>
<th>Relative no. of comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>KB</td>
<td>184</td>
<td>1.71</td>
<td>5.67</td>
<td>105</td>
<td>6</td>
<td>103.7%</td>
</tr>
<tr>
<td>real</td>
<td>241</td>
<td>1.91</td>
<td>6.38</td>
<td>124</td>
<td>6</td>
<td>100.3%</td>
</tr>
<tr>
<td>KB</td>
<td>269</td>
<td>1.72</td>
<td>5.16</td>
<td>164</td>
<td>7</td>
<td>107.9%</td>
</tr>
<tr>
<td>KB</td>
<td>320</td>
<td>1.85</td>
<td>8.13</td>
<td>141</td>
<td>12</td>
<td>110.0%</td>
</tr>
<tr>
<td>random</td>
<td>228</td>
<td>2.38</td>
<td>8.88</td>
<td>142</td>
<td>8</td>
<td>115.7%</td>
</tr>
<tr>
<td>mod.</td>
<td>563</td>
<td>2.58</td>
<td>12.24</td>
<td>338</td>
<td>7</td>
<td>114.5%</td>
</tr>
<tr>
<td>KB</td>
<td>1263</td>
<td>3.18</td>
<td>16.61</td>
<td>661</td>
<td>11</td>
<td>108.9%</td>
</tr>
<tr>
<td>KB</td>
<td>2389</td>
<td>3.50</td>
<td>25.49</td>
<td>1188</td>
<td>10</td>
<td>111.3%</td>
</tr>
<tr>
<td>KB</td>
<td>3658</td>
<td>3.82</td>
<td>27.20</td>
<td>1703</td>
<td>8</td>
<td>105.3%</td>
</tr>
<tr>
<td>KB</td>
<td>3905</td>
<td>4.04</td>
<td>33.95</td>
<td>1858</td>
<td>11</td>
<td>99.9%</td>
</tr>
<tr>
<td>KB</td>
<td>301</td>
<td>7.67</td>
<td>42.11</td>
<td>88</td>
<td>8</td>
<td>13.2%</td>
</tr>
<tr>
<td>KB</td>
<td>301</td>
<td>6.40</td>
<td>10.43</td>
<td>168</td>
<td>6</td>
<td>102.7%</td>
</tr>
<tr>
<td>KB</td>
<td>301</td>
<td>4.22</td>
<td>5.68</td>
<td>205</td>
<td>4</td>
<td>101.3%</td>
</tr>
<tr>
<td>KB</td>
<td>586</td>
<td>9.93</td>
<td>72.55</td>
<td>144</td>
<td>9</td>
<td>67.2%</td>
</tr>
<tr>
<td>KB</td>
<td>586</td>
<td>7.72</td>
<td>11.50</td>
<td>353</td>
<td>5</td>
<td>102.9%</td>
</tr>
<tr>
<td>KB</td>
<td>995</td>
<td>5.52</td>
<td>250.24</td>
<td>85</td>
<td>28</td>
<td>16.2%</td>
</tr>
<tr>
<td>KB</td>
<td>995</td>
<td>16.40</td>
<td>62.88</td>
<td>354</td>
<td>9</td>
<td>91.4%</td>
</tr>
<tr>
<td>KB</td>
<td>995</td>
<td>13.58</td>
<td>28.38</td>
<td>506</td>
<td>6</td>
<td>105.1%</td>
</tr>
<tr>
<td>KB</td>
<td>1266</td>
<td>7.58</td>
<td>321.19</td>
<td>100</td>
<td>30</td>
<td>12.9%</td>
</tr>
<tr>
<td>KB</td>
<td>1266</td>
<td>18.22</td>
<td>76.87</td>
<td>438</td>
<td>9</td>
<td>89.8%</td>
</tr>
<tr>
<td>KB</td>
<td>1266</td>
<td>17.82</td>
<td>41.70</td>
<td>592</td>
<td>6</td>
<td>101.1%</td>
</tr>
</tbody>
</table>

### Exploiting the Concept Structure

1. If a concept is defined conjunctively, i.e.,

\[ A \equiv X \land Y \land \ldots \land Z \]

all atomic concepts \((X, Y, Z)\) are obviously super-concepts. So one can mark these concepts as subsuming (and can propagate this information).

2. Primitive concepts (introduced with \(\Box\)) can be ignored in the bottom search phase, if we classify in definition order (i.e., before a term is used, it will be defined).

3. **Primitive components** of a concept are the undefined concepts in the unfold form of a concept. A subsumption test between two conjunctively defined concepts can only be successful if the primitive components are in an set-inclusion relationship.

### Caching and Lazy Unfolding

- Instead of unfolding a concept and then testing subsumption, one can **delay the expansion** of a defined concept until no other way to prove unsatisfiability is possible.
- Instead of testing for clashes of primitive concepts, we also look for clashes on the level of named, defined concepts.
- Finally, we can in addition also look for clashes implied by the subsumption hierarchy, e.g., \(\text{Man} \land \lnot \text{Human} \) form a clash.

~~ These optimization saved roughly 50% of the runtime on the realistic KBs.

### Provisional Balance

- The described methods (and a few others) were used in speeding up the KRIS system.
- We achieved a speedup factor of around 400.
- The difference between a first prototype implementation that just implements the principles and a highly optimized system can be huge – in particular when we are dealing with worst-case exponential time/space problems.
- Interestingly, the method we believed to be superior (chain insertion) was not effective at all.
- One has to look for what typical cases require – and these cases might not be easily recognizable as polynomial special cases.
- There are techniques especially geared towards more expressive DLs – and some we will have a short look at.
Semantic Branching in Tableau Proofs

Consider the System: \( S = \{ x : (A \cup B), x : (A \cup C) \} \)

Ordinarily,
- we would first try out \( A \) and \( B \), nondeterministically.
- Then we would make another nondeterministic choice between \( A \) and \( C \).

if \( A \) leads to clash, we would try that out twice.

Use semantic branching instead (similar to what is done in the Davis-Putnam procedure).

This avoids double work on \( A \) and leads to tighter constraints (when using \( \neg A \) in the other branch).

Experience shows that the additional overhead for propagating \( \neg A \) is usually negligible

Local simplification becomes possible

Boolean Constraint Propagation

When using semantic branching, we can also simplify disjunctions by propagating the concepts, we have branched on, using Boolean constraint propagation (aka unit propagation).

Example:

\[ S = \{ x : (C \cup (D_1 \sqcap D_2)), x : (\neg D_1 \sqcap \neg D_2 \sqcup C), x : \neg C \} \]

Because of \( x : \neg C \), this can be simplified to

\[ S = \{ x : (D_1 \sqcap D_2), x : (\neg D_1 \sqcup \neg D_2 \sqcup C), x : \neg C \} \]

This in turn can be simplified to

\[ S = \{ x : (D_1 \sqcap D_2), x : C, x : \neg C \} \]

This is easily be detected as unsatisfiable!

Note that no nondeterministic branching was necessary!
Absorption

- Since general inclusion statements lead to nondeterminism, it is a good idea to reduce these as much as possible.
- Use the following equivalences to absorb general inclusion statements into primitive concept definitions:

\[ \begin{align*}
C_1 \sqcap C_2 \sqsubseteq D & \iff C_1 \sqsubseteq D \sqcap \neg C_2 \\
C \sqsubseteq D_1 \sqcap D_2 & \iff (C \sqsubseteq D_1), (C \sqsubseteq D_2)
\end{align*} \]

Example Absorption. Assume: Geometric-figure $\sqsubseteq$ Figure, Geometric-figure $\sqcap \exists$angles.Three $\sqsubseteq \exists$ sides.Three.

- The latter inclusion statement can be massaged into:

Geometric-figure $\sqsubseteq \exists$ sides.Three $\sqcap \neg \exists$ angles.Three

- Then it can be absorbed into:

Geometric-figure $\sqsubseteq$ Figure $\sqcap \exists$ sides.Three $\sqcap \neg \exists$ angles.Three

- Using this technique can make a huge difference

Conclusion

- Implementing inference methods for DLs is easy – if we do not care about efficiency
- The difference between a prototype implementation and a carefully optimized system can be several orders of magnitude

Optimizations are possible on classification level, the subsumption testing level, and the tableau proof level

- These optimization should be geared towards the typical case
- It is nearly impossible to decide analytically, which method is the right one, only empirical comparisons can help here

Note: Although there exists now a number of efficient DL methods, in general there will remain the problem that sometimes these systems just have to give up because a subsumption query is to difficult!

Literature